AD

TECHNICAL REPORT ECOM-01901-26

GRAPHICAL-DATA-PROCESSING RESEARCH STUDY AND EXPERIMENTAL INVESTIGATION

FOURTH QUARTERLY REPORT

By

R. O. Duda *P. E. Hart* *J. H. Munson*

MARCH 1967

DISTRIBUTION STATEMENT

Distribution of this document is unlimited.

ECOM

UNITED STATES ARMY ELECTRONICS COMMAND · FORT MONMOUTH, N.J.

CONTRACT DA 28-043 AMC-01901(E)

STANFORD RESEARCH INSTITUTE

Menlo Park, California

TECHNICAL REPORT ECOM-01901-26 MARCH 1967

GRAPHICAL-DATA-PROCESSING RESEARCH STUDY AND EXPERIMENTAL INVESTIGATION

REPORT NO. 26, FOURTH QUARTERLY REPORT

1 NOVEMBER 1966 TO 31 JANUARY 1967

SRI Project 5864

CONTRACT NO. DA 28-043 AMC-01901(E)

Continuation of Contract No. DA 36-039 AMC-03247(E)

Task No. 1P6-20501 A-448-02-45

Prepared By

R. O. DUDA P. E. HART J. H. MUNSON

STANFORD RESEARCH INSTITUTE
MENLO PARK, CALIFORNIA

For

U.S. ARMY ELECTRONICS COMMAND, FORT MONMOUTH, N.J. 07703

ABSTRACT

This report describes the continuing development of preprocessing, classification, and context analysis techniques for hand-printed text, which are advancing at an accelerating pace.

Experiments have been continued with the Piecewise-Linear learning machine, using the outputs of two preprocessors: the PREP 24A simulation of the 1024-image optical preprocessor, and the CALMMASK preprocessor, which employs both edge-detecting and corner-detecting masks. A new low test error rate for classification has been achieved on hand-printed alphabets of FORTRAN characters.

Statistics of the performance of the learning machine during a single testing iteration are presented, and shed light on several important questions, such as the distribution of rankings of the desired character category when it is not in first place.

A discussion of the preprocessing methods used in the topological approach to preprocessing and classification is begun.

The initial development of a FORTRAN syntax analyzer is described. A milestone has been reached with the passage of a small sample of actual FORTRAN text from a coding sheet through the scanning, preprocessing, classification, and syntax-analysis programs.

CONTENTS

CONTENTS (Concluded)

ILLUSTRATIONS

TABLES

BLANK PAGE

I INTRODUCTION

The development of preprocessing, classification, and context-analysis techniques for hand-printed text is progressing at an accelerating pace.

Section II of this report describes experiments with two template-matching preprocessors and the CALM simulation of the Piecewise-Linear Learning Machine. One series continues the nine-view experiments with the outputs of the PREP 24A simulation of the 1024-image optical preprocessor. A new low test error rate is achieved as the training set is expanded. In a new series of experiments, CALM is used on the outputs of a different simulated preprocessor (the CALMMASK program), utilizing both edge-detecting and corner-detecting masks, together with the features derived from Clemens' technique.

In an interesting new line of data analysis, detailed statistics were developed for the performance of the learning machine during a single testing iteration. These are discussed in Sec. II. These statistics shed light on several questions important to both the classifier and the context analyzer--for example, the distribution of rankings of the desired character category when it is not in first place.

A discussion of the preprocessing methods used in the topological approach to preprocessing and classification (formerly called the AD HOC approach) is begun in Sec. III. An improved classification routine is being developed, and it is planned that extensive further discussion of these methods will be presented in the next report.

The initial development of a FORTRAN syntax analyzer, which is the heart of the context analyzer, is described in Sec. IV. A milestone has been reached with the passage of a small sample of actual FORTRAN text from a coding sheet through the scanning, preprocessing, classification,

and syntax-analysis programs. The results of this experiment (presented in Sec. IV) indicate the power of the syntax analysis in cleaning up text with misclassified characters.

II EXPERIMENT WITH TWO TEMPLATE-MATCHING PREPROCESSORS AND THE PIECEWISE-LINEAR LEARNING MACHINE

A. Further Experiments with the Edge-Detecting Preprocessor and the Piecewise-Linear Learning Machine

We have continued the series of experiments described in the Second and Third Quarterly Reports with two additional experiments. The basic feature vectors used in these experiments, as in the ones described previously, were the nine-view, 84-bit binary vectors produced by the PREP 24A simulation of the 1024-image optical preprocessor. Each of the 84 bits specifies the detection of an edge of a certain orientation in a certain region of the image field. The classifier used, as before, was the CALM (Collected Algorithms for Learning Machines) simulation of the 46-category Piecewise-Linear Learning Machine, with two dot product units per category.

1. PREP-CALM Experiment 8

This was a re-run of Experiment 3 (Described in Sec. II of the Second Quarterly Report), in which we added to the feature vectors the 24 bits generated by Clemens' technique (also described in the Second Quarterly Report). The patterns used in Experiment 3 were 9-view, 84-bit patterns. In Experiment 8, each of the single-view patterns was augmented by the addition of 24 bits, broken into eight segments of three bits each. Within each of the eight segments, the three bits were used to encode the number of occurrences of extrema of the figure boundary (in X or Y) in one of the four quadrants of the figure.

The result of combining the edge-detection data with the Clemens' technique data was thus a set of nine feature vectors (views) for each character. The feature vectors each had 108 (84 + 24) components. The 24 Clemens' bits were the same throughout all nine views, whereas the edge-detection bits varied.

The results of the learning-machine experiment on this set of feature vectors are presented in Fig. 1. The training error rate decreased to 23.4% in five iterations. The one-view independent test error rate dipped to 33.9%, then rose to 39.9% at Iteration 4. The nine-view test error rate was calculated at Iteration 3 (22.6%) and at Iteration 5 (23.7%).

(The graphs of Fig. 1 and similar figures are prepared by a separate small program for the SDS 910 computer, called ERROR GRAPH. The training error rate is the generally lower curve, whose numerical values are listed below the curve. The test error rate is the other curve. The precision of plotting the ordinate values is limited to half-line spacing vertically by the use of the computer's typewriter for preparing the graph; each vertical half-space corresponds to 1.2 or 1.3%.)

The results of Experiment 8 may be compared with those of Experiment 3, in which the training error rate reached 36% in 5 iterations, the one-view test error rate reached 45%, and the nine-view test error rate was 23% (Fig. 2). We see that the addition of the Clemens' technique bits in the present experiment has considerably improved the one-view training and test error rates during the first five iterations, but has had essentially no effect on the important nine-view test error rate. This result would appear to reflect the fact that the new bits, while contributing valuable information to each view, do not contribute correspondingly to the majority-voting nine-view recognition process, because the bits are the same in all views. The improvement in recognition rate using nine views may be thought of as resulting from the outvoting of a "bad" view or views by the others, and this cannot happen when the information in all views is the same.

In conclusion, we found that the extra information carried in the Clemens' technique bits did improve the single-view classification of patterns, but that without a "9-view generalization" of the Clemens' technique the improvement did not carry over noticeably to 9-view classification.

LEARNING CURVES

SRI PROJECT 5864, EXPT NO. 8
RUN ON 84-BIT 9-VIEW PATTERNS PLUS CLEMENS BITS. TRAIN AUTHORS 1-12, TEST13-16.
THERE ARE 1656 TRAINING PATTERNS AND 552 TESTING PATTERNS

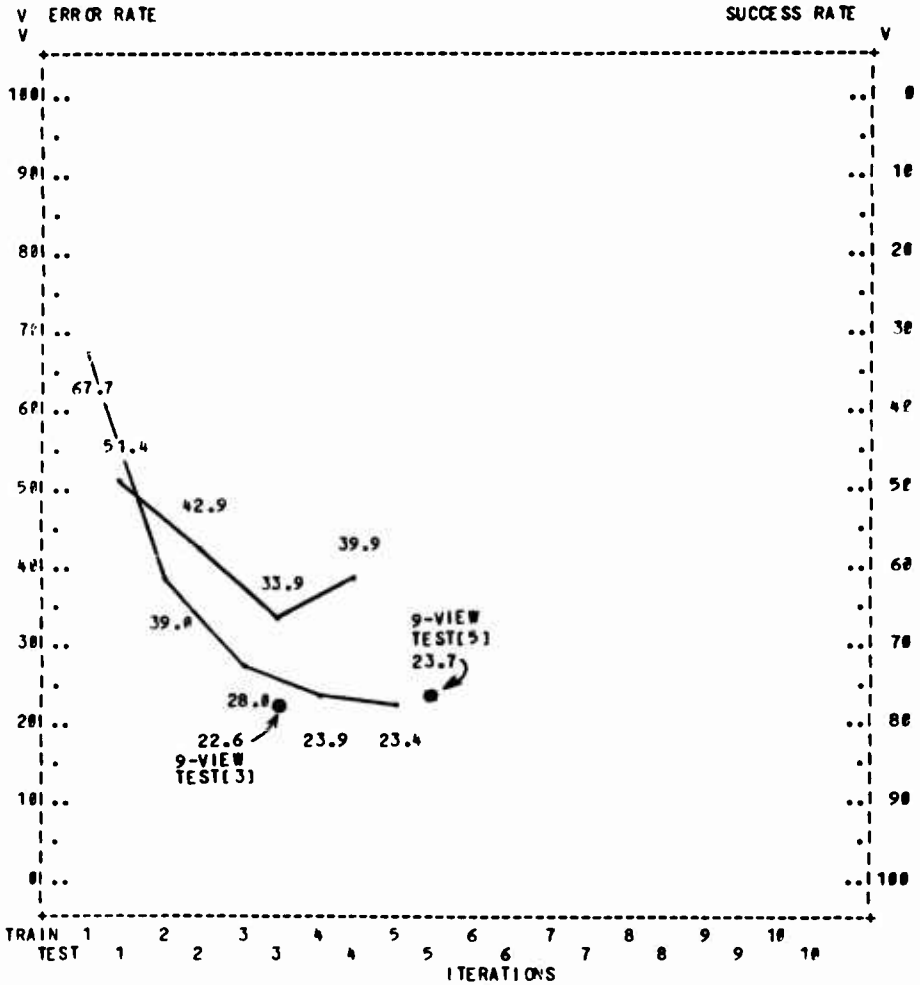

FIG. 1 PREP-CALM EXPERIMENT 8

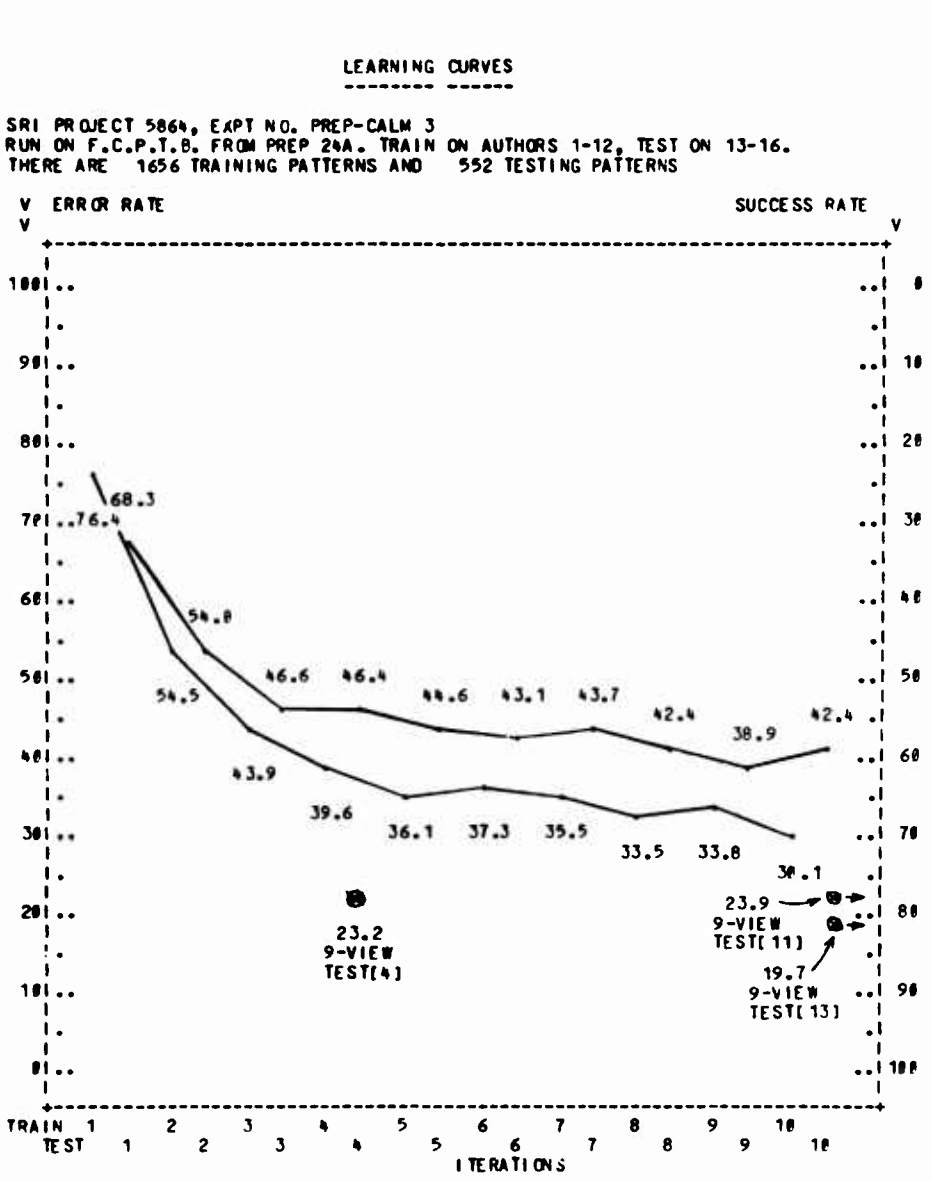

FIG. 2 PREP–CALM EXPERIMENT 3

2. PREP-CALM Experiment 9

Experiment 9 was a re-run of Experiment 3 with an expanded training set. The patterns used were the same as those in Experiment 3: 84-bit, nine-view patterns. This time, in addition to the three FORTRAN alphabets from each of twelve authors for training and four authors for testing, three alphabets from each of eight more training authors had been preprocessed through PREP 24A.

The results for Experiment 9 are presented in Fig. 3. The training error rate decreased to 31.4% at Iteration 10. The one-view test error rate ranged between 38 and 42% from Iteration 3 through Iteration 10. The nine-view test error rate was 18.8% at Iteration 5 and 19.6% at Iteration 10. These values represent a new low in test error rate for the FORTRAN characters, and, apart from statistical fluctuations, appear to be a couple of percent lower than the results of Experiment 3. It may be noted that the training and test error curves are quite close together, indicating that the expanded training set is largely successful in representing the test data.

B. Experiments with the CALMMASK Preprocessor and the Piecewise-Linear Learning Machine

1. The CALMMASK Preprocessing Program

As an aid to the development of new templates (or masks) for preprocessing, and new structures combining these templates, a program called CALMMASK was written for the SDS 910. CALMMASK implements simulated optical masks of the type used in the 1024-image preprocessor and the PREP 25A simulation--for example, edge-detectors. CALMMASK allows additional flexibility in the use of these templates. The shapes and threshold values of individual template types may be specified; several templates of the same or different types may then be combined by logical OR-ing and AND-ing into a feature; and features may be replicated at various locations on the pattern image field. Parameters controlling all of these options are under direct control from the computer console.

LEARNING CURVES

SRI PROJECT 5864, EXPT NO. PREP-CALM 9
RUN ON F.C.P.T.B. FROM PREP 24A. TRAIN ON AUTHORS 1-12,17-24, TEST ON 13-16.
THERE ARE 2760 TRAINING PATTERNS AND 552 TESTING PATTERNS

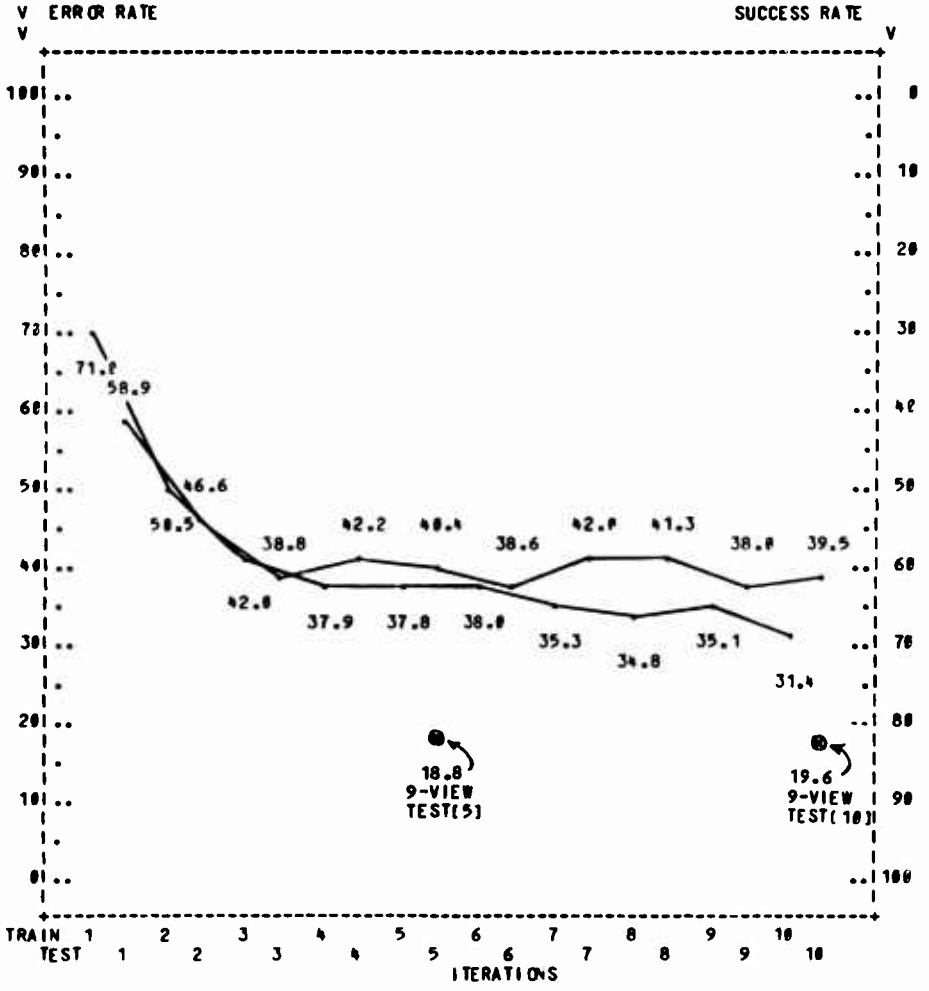

FIG. 3 PREP-CALM EXPERIMENT 9

CALMMASK exists in two versions. The "interactive" version allows an experiment to design features on-line by specifying them at the console, observing their behavior when presented with test patterns, and modifying them at will. The "production" version provides a more efficient program for processing large quantities of patterns through an already-designed preprocessor.

The CALMMASK feature set used in the experiments to be described here was as follows: There were 16 types of template. Twelve of these were edge detectors similar to those employed in PREP 24A, oriented at each 30^0 of the compass. The remaining four were corner detectors, designed to detect the corners formed by the meeting of a vertical and a horizontal stroke (Fig. 4). Each corner template had 14 cells with a

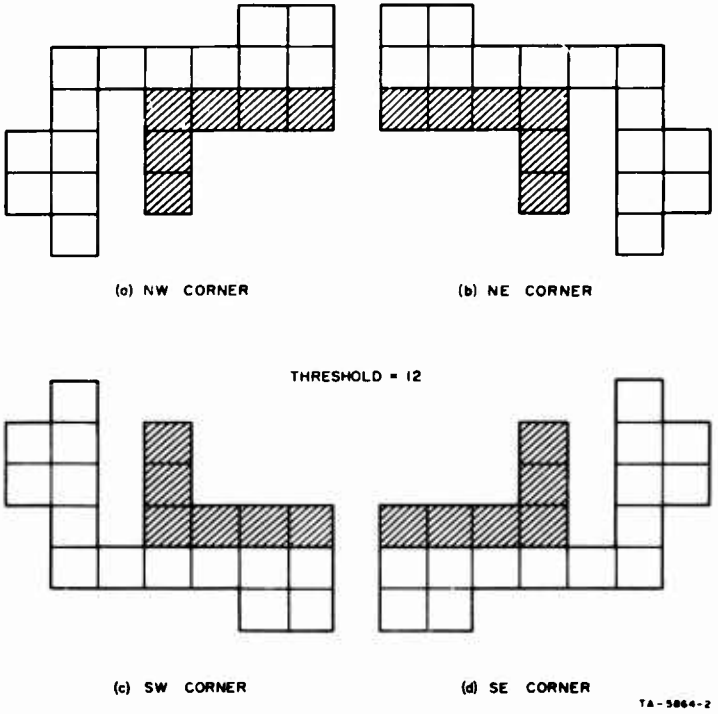

FIG. 4 CORNER-DETECTING TEMPLATES

weight of +1, 6 cells with a weight of -1, and a threshold of 12. It may be noted that the templates are more tolerant of the orientation of the vertical stroke than of the horizontal, reflecting the characteristics of actual printing.

Each of the 16 template types was placed in each of the four quadrants of the image field, giving a total of 64 pattern components (features) in the output of CALMMASK. Within each quadrant the template was presented in every vertical and horizontal location, and a response from the template in any location caused a positive response for the corresponding feature. (In other words, each feature was a many-way OR function of all the responses for the locations throughout the quadrant.)

The patterns were not translated before presentation to the templates, as was the case with PREP 24A; thus, only one-view feature vectors were obtained from CALMMASK. It was expected that the presentation of the templates in every location would have much the same effect as the translation of the patterns to give the nine-view PREP 24A feature sets. One purpose of the experiments was to compare the two approaches to translation invariance; the other purpose was to see the effect of the corner-detecting templates.

2. MASK-CALM Experiment 2

Following a shakedown experiment, a full set of patterns was preprocessed with CALMMASK and presented to the CALM simulation of the Piecewise-Linear learning machine. The 24 Clemens' technique bits described above were added to the patterns as they were presented to CALM, forming feature vectors of 88 (64 + 24) bits. In this MASK-CALM Experiment 2 the training and testing sets were the same as in Experiment 3 of the previous series, which used patterns preprocessed by PREP 24A. Thus, a direct comparison is possible. The training set consisted of three FORTRAN alphabets from each of twelve authors; and the test set, of three alphabets from each of four authors.

Figure 5 shows the results of the experiment The training error rate decreased to 2 9% in five iterations. The training error rate during PREP-CALM Experiment 3 never improved beyond 30% (however, it must be remembered that only one-view patterns were used in the present instance, so the identical feature vector was presented at each iteration, forming an easier training problem). The test error rate decreased to 25.2%, then rose to 27.0% at Iteration 5. These rates may be compared with the 23% test error rate of PREP-CALM Experiment 3 at Iteration 4. The single-view pattern vectors from CALMMASK performed almost as well as the nine-view vectors from PREP 24A.

3. MASK-CALM Experiment 3

In MASK-CALM Experiment 3, six more authors (18 alphabets) were added to the training set. Other details stayed the same as in the previous experiment. As shown in Fig. 6, the training error rate decreased to 8.1% in four iterations, and the test error rate reached 24.6%.

4. MASK-CALM Experiment 4

In Experiment 4, the first six authors (18 alphabets) in the pattern file were used for testing, and the seventh through twenty-second authors for training. Again, all other details of the experiment were the same as in the two previous experiments. The experiment was carried for ten iterations to check for any extra long-term improvement in the test error rate. Figure 7 shows that the test error rate flattened out at 22 to 23% after the third iteration. The training rate reached 3.5%.

Comparison of Experiments 3 and 4 with Experiment 2 show that the increased training set has improved the test error rate on the CALMMASK patterns by a small amount. To date, the best error rate in the experiments on the CALMMASK patterns (22% in Experiment 4) has not matched the best 9-view rate (19% in Experiment PREP-CALM 9).

5. MASK-CALM Experiment 5

Experiment 5 was performed to isolate the effect of the Clemens' technique bits, which had been included with the template features throughout the other MASK-CALM experiments. In Experiment 5, only the

LEARNING CURVES

SRI PROJECT 5864, EXPT NO. MASK-CALM 2
RUN ON 88-BIT PATTERNS FROM CALMMASK + CLEMENS. 12 TRAINING AUTHORS, 4 TEST
THERE ARE 1656 TRAINING PATTERNS AND 552 TESTING PATTERNS

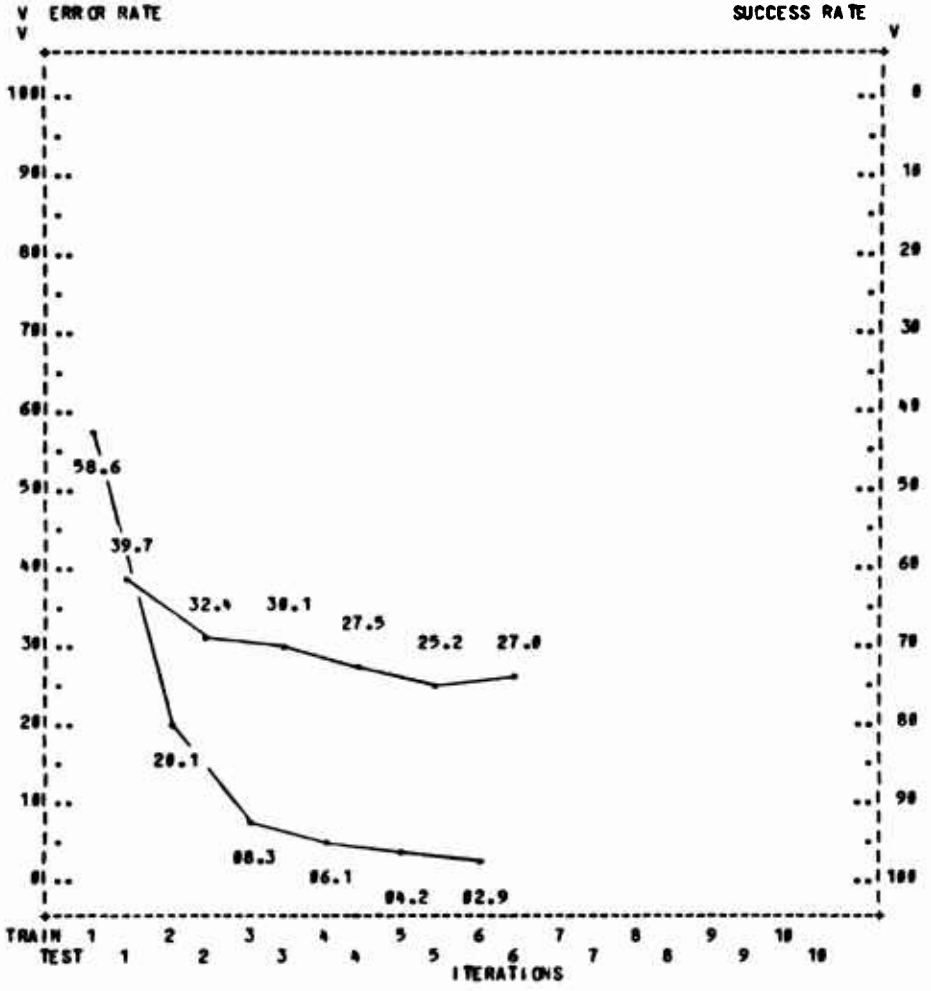

FIG. 5 MASK-CALM EXPERIMENT 2

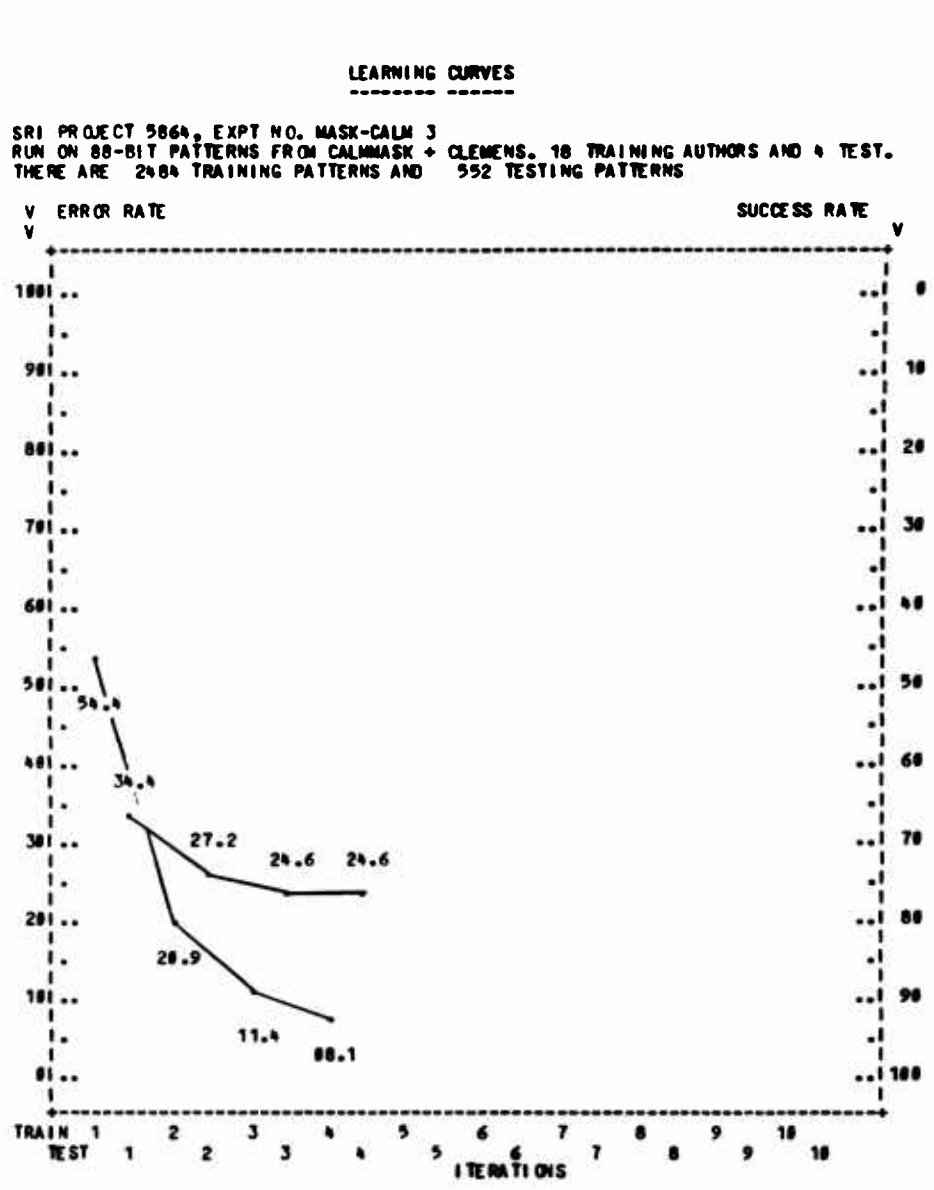

FIG. 6 MASK-CALM EXPERIMENT 3

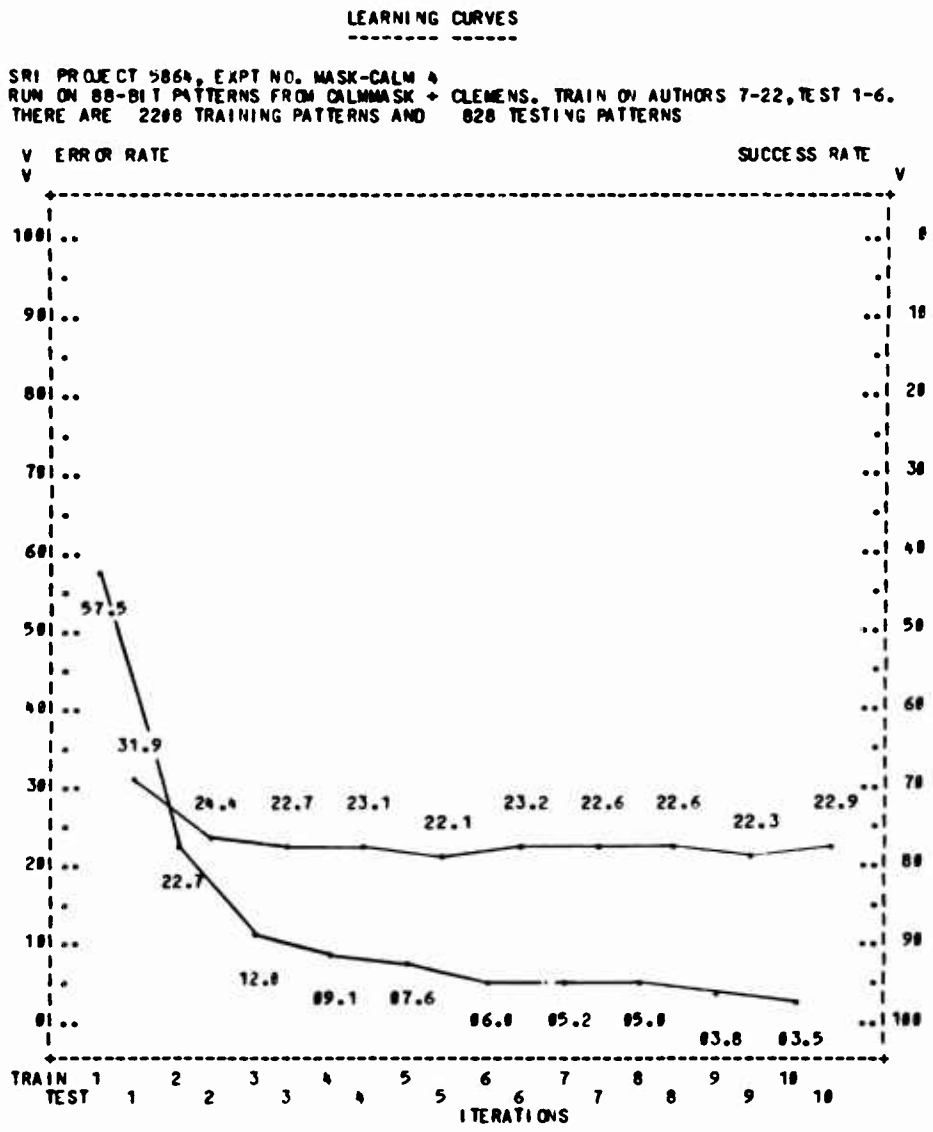

FIG. 7 MASK-CALM EXPERIMENT 4

64 template feature bits were used. The training and testing sets were the same as in Experiment 4. Figure 8 shows that the error rates were increased by the deletion of Clemens' technique bits: in four iterations, the training error rate reached 14.5% and the test error rate reached 32.2%.

C. Examination of Learning-Machine Statistics During a Test Iteration

A small modification was made to the CALM program, which allowed certain statistics concerning the performance of the Piecewise-Linear learning machine to be gathered during the running of an iteration. Raw statistics gathered included the values of the largest and second largest category responses (Dot Product Unit sums) for each pattern, the ranking of the desired (true) category, and its sufficiency or deficiency.

The ranking of the desired category can range from 1 to 46. It is 1 for a pattern if and only if the pattern is correctly classified. If the ranking is 1, the sufficiency is defined as the difference between the DPU sum of the desired category and the largest of the other sums (which will belong to the second-ranked category). If the pattern is in error, the deficiency is defined as the difference between the largest sum (the one in the chosen category) and the sum for the desired category.

The sufficiency (or deficiency) measures the closeness of the machine's decision, and thus can be interpreted as a measure of confidence in the category chosen. (If the ranking of the desired category is 3 or greater, the deficiency does not show how close the second choice was to the first choice, but this is a small point.)

Since the significance of individual DPU sums is clearer in a one-view than in a nine-view experiment, we chose a one-view test iteration for analysis: Test Iteration 4, from MASK-CALM Experiment 4. Figure 7 shows the error rate for this iteration to be 23.1%.

The distribution of rankings of the desired category is shown in Table I. Rankings 1 and 2 include the correct category almost 90% of the time; rankings 1 through 4 include the correct category 95% of the time.

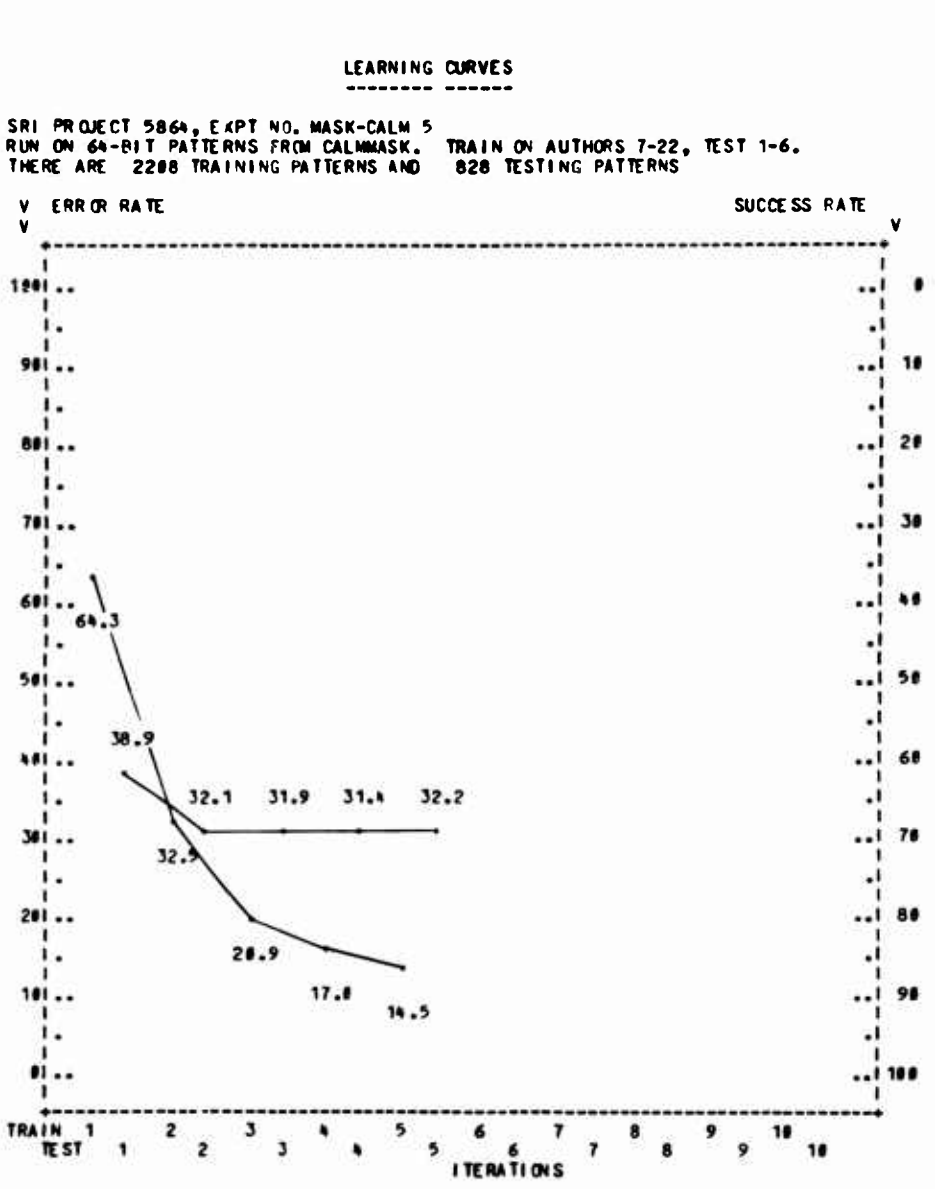

FIG. 8 MASK-CALM EXPERIMENT 5

Table I

MASK-CALM EXPERIMENT 4, TEST ITERATION 4--
RANKINGS OF DESIRED CATEGORY

Ranking	Occurrence	%	Cumulative %
1	637	76.9	76.9
2	96	11.6	88.5
3	40	4.8	93.3
4	14	1.7	95.0
5	9	1.1	96.1
6	7	0.8	96.9
7	2	0.2	97.1
8	3	0.4	97.5
9	1	0.1	97.6
10	5	0.6	98.2
12	2	0.3	98.5
13	3	0.4	98.9
14	1	0.1	99.0
17	2	0.3	99.3
19	5	0.6	99.9
27	1	0.1	100.0
TOTAL	828	100.0	100.0

Thus, presenting the first few choices to the context analyzer leads to a very high probability of including the correct category.

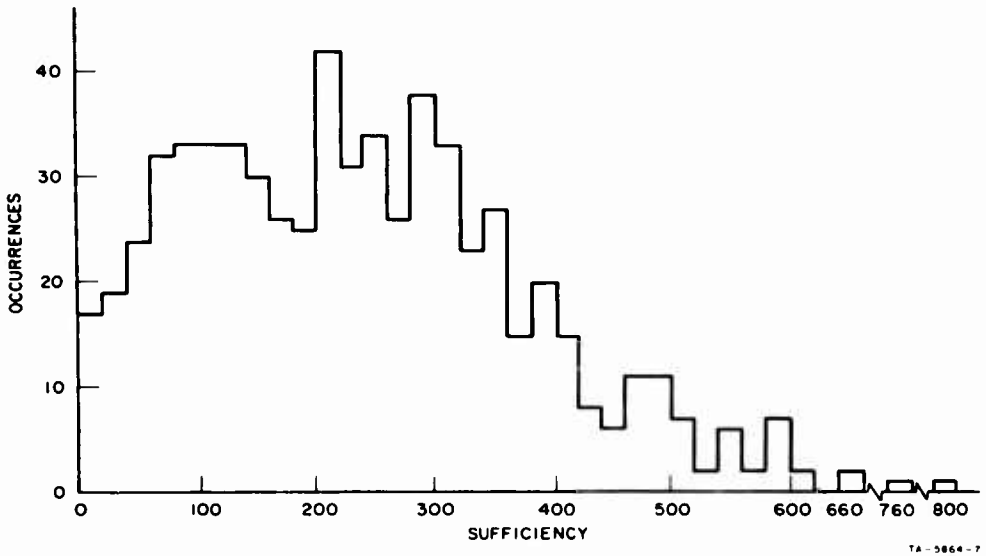

FIG. 9 HISTOGRAM OF SUFFICIENCIES

Figure 9 is a histogram of the sufficiencies of the correctly classified patterns. Figure 10 is a histogram of the deficiencies of the incorrectly classified patterns, broken down according to ranking. Figure 11 is a histogram of values of the maximum DPU sum formed for every pattern, broken into two parts: for the correctly classified patterns, and for the patterns in error. A number of interesting conclusions can be drawn from these graphs.

The first, and quite surprising fact to be observed from the histograms is the great range of the maximum DPU sums (from approximately 280 to 1310), sufficiencies (up to 800), and deficiencies (up to 600). The CALM program records and prints out the overall maximum DPU sum formed during an entire iteration, as a check against overflow in the computer.

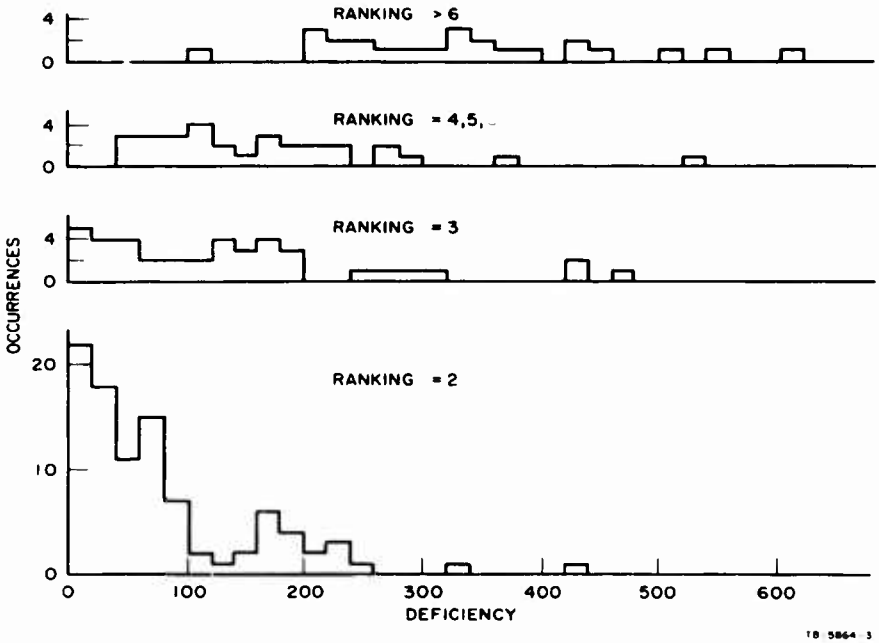

FIG. 10 HISTOGRAMS OF DEFICIENCIES vs. RANKING

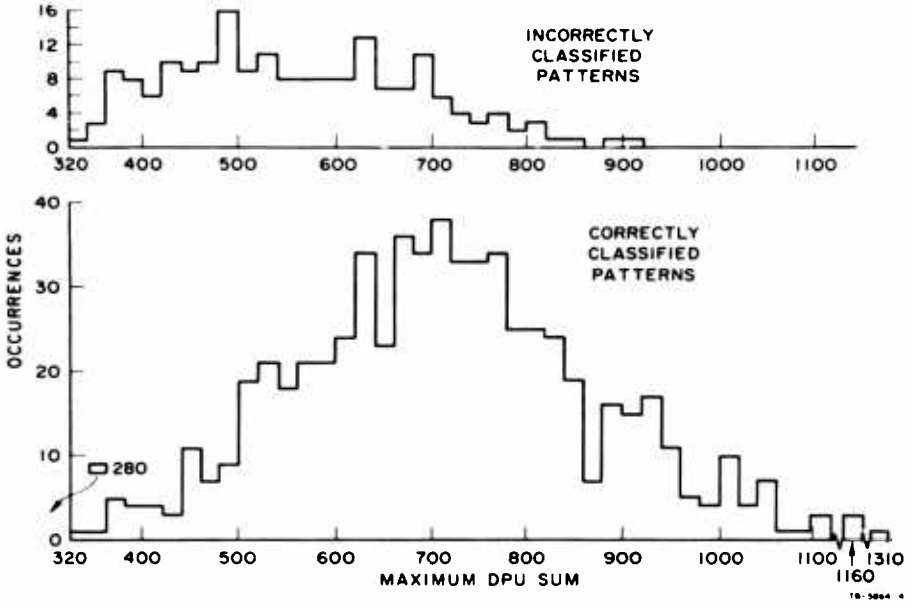

FIG. 11 HISTOGRAMS OF MAXIMUM DPU SUMS

In this case, the largest sum was 1312, and this figure is in the typical range for experiments run with CAIM. We may assume that the most negative DPU sum formed during the iteration was comparable in magnitude. This means that all of the DPU sums formed for each pattern lie in an interval of length approximately 2500.

If the 46 category responses in the machine were randomly distributed in the range -1200 to 1300, the average numerical interval between responses would be about 50. Even with fluctuations, we would expect the sufficiencies, most of the deficiencies, and the variation in maximum sums all to range up to only 100 or 200. Yet we find spreads of 600 to 1000, and occurrences such as a pattern for which not one of the DPU sums exceeded (approximately) 280. Such behavior is quite contrary to our intuition, which expected much tighter distributions of these quantities. Since the actual performance of the learning machine is an a priori fact, we do not infer that the observed distributions are in themselves "good" or "bad"--merely surprising.

A second observation is that the distribution of maximum DPU sums is higher, on the average, when the pattern is correctly classified than when it is not. It might be possible to use the maximum sum to adjust the confidence measures of the chosen and competing categories, unless the maximum sum is so correlated with the sufficiency and deficiency that there is little or no independent information to be gained.

Turning to the histogram of sufficiencies, we find a tendency, which appears to be statistically significant, for depletion in the region near zero. Since the sufficiency and deficiency are measures of the same quantity (namely, desired-category response minus the maximum of other responses), we can further study this effect by combining the deficiency histograms of Fig. 10, reversing the horizontal axis, and placing the resulting histogram beside that of Fig. 9. This is done in Fig. 12. It is evident that the dropoff in sufficiencies near zero is related to the continuing dropoff of occurrences with increasing deficiency. (Since there is no reason to expect a discontinuity at zero, the jump there is probably a statistical fluctuation.)

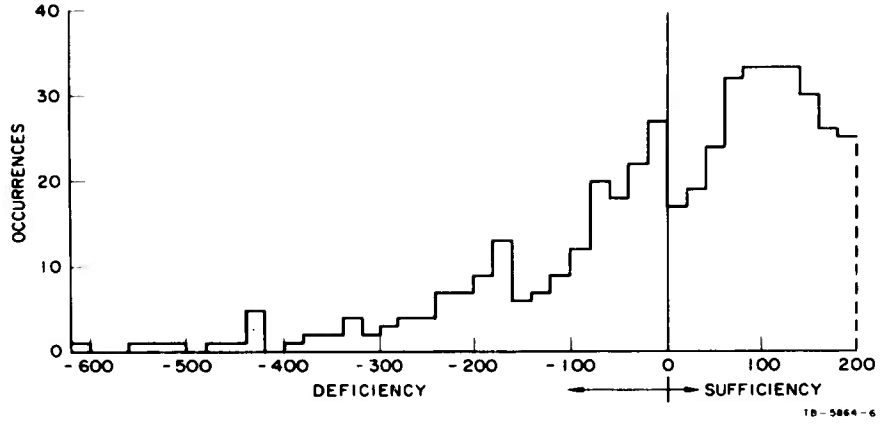

FIG. 12 COMBINED HISTOGRAMS OF DEFICIENCIES AND SUFFICIENCIES

During MASK-CALM Experiment 4, the training margin was set to 88 (the number of pattern components). This value closely matches the value of sufficiency at which the dropoff occurs. It is an attractive hypothesis that the margin has tended to "push" sufficiencies above the 88 level, although this one example is only limited evidence. If the hypothesis is true, the results are quite satisfying, because although the training governed by the training margin was applied only to the training patterns, we here see its effect in enhancing the decisions on the test patterns.

Finally, the study of information such as that in Fig. 10 will be of value in the future, when more is known about the needs of the context analyzer. Figure 10 portrays the relation between the ranking of a misclassified character and its deficiency. A study of relations such as this will indicate, for example, how much weight should be given to the ranking, and how much to the differences in DPU sums, when determining the confidence measures to be assigned to each category.

III TOPOLOGICAL PREPROCESSING OPERATIONS FOR HANDPRINTED CHARACTER RECOGNITION

A. Introduction

Let us propose, with the usual risk of oversimplification, the following difference among methods of extracting feature information for the recognition of graphical patterns such as handprinted characters. On one hand are the "topological" preprocessing methods; on the other hand, the "non-topological" ones.

The topological methods extract from the character image those types of features that would be commonly used by people asked to describe the shapes of characters. Typical descriptions are: "A letter P has a closed loop on top, with a stroke sticking down from it--on the left-hand side." "The difference between a letter O and a letter D is that the O is round, but the D has two corners on the left," and so on. Topological features include strokes, loops, hollows, corners, curvatures, connections, etc., as well as the relative positions and orientations of the basic features. In short, these features are primarily concerned with the geometrical and topological components and relationships of the character as a whole.

We may characterize the non-topological methods, by contrast, as those which derive information less related to the "natural" or intuitive description of the character at the topological level. Clemens' technique (described in Ref. 1* and in the Second Quarterly Report), in which the x and y extrema of the contour of the character are recorded, is an example of such a method. Integral geometry (Quarterly Reports 3 and 4 of the preceding Contract No. DA 36-039 SC-78343), in which statistical measurements are made of the intersections of a pattern with randomly chosen lines, is a prime example of a method seemingly unrelated to the natural description of the character. The character-recognition literature provides many more examples of non-topological feature-extraction

* References are listed at the end of the report.

techniques, such as random sampling (e.g., Perceptron and N-tuple) methods,[2] and the sequence of intersections of the character with a scan line of fixed orientation.*[3] Finally, in this framework, the extraction of features by edge-detecting masks (as exemplified by the 1024-image preprocessor) falls in the non-topological category.

Non-topological preprocessing methods are often prompted by their elegance and simplicity, and the convenience of a uniform approach. Most such techniques are based on elegant or "clever" processes that are quite simple conceptually, and that are correspondingly easy and straightforward to implement in a computer program or in hardware. If a process generates sufficient information to allow unique classification of well-formed characters, it becomes a candidate for a preprocessing technique. The major problem that confronts such methods arises when they are faced with the ill-formed characters that do occur in actual input and must be handled. The method based on a single organizing principle often seems to lack the "ruggedness" to maintain the constancy of its outputs in the face of character distortions and aberrations, and no corrective recourse is available within the framework of the single uniform approach.

The topological preprocessing methods gain their appeal from the fact that they use the same features used by humans in describing the characters. It can then be hoped that when faced with distortions that leave a character still recognizable by humans, such methods will preserve information sufficient for classification. As a corollary, human introspection together with observation of the system's operation can be used as guides for designing, evaluating, and improving the preprocessor and subsequent classifier.

B. Current Status of the Topological Preprocessing and Classification Program

A preliminary program for the preprocessing of handprinted characters by the extraction of topological features was described in the third

* The "Birdwatch" technique, developed by Rabinow Engineering Co., described in Ref. 1.

Quarterly Report under the heading, "AD HOC Preprocessing and Classification of Characters." The preliminary program contained routines for finding the connected components of a character image, its boundary, convex hull, enclosures, concavities of the boundary, and spurs (strokes that end at an isolated tip). The preliminary program consisted almost entirely of these preprocessing routines. Only a fragmentary classification routine, with a decision tree for handling single-stroke characters, had been added.

An extensively modified version of this program, called TOPO 2, is currently being written. Changes of three types are being made in TOPO 2. First, needed improvements have been introduced into the boundary-following and stroke-tracing routines. Second, a general cleanup of the coding was undertaken, primarily to reduce running time and storage requirements. Third, the decision tree approach to classification that had been begun in the AD HOC program has been dropped in favor of producing alternative classifications with confidence measures.

The change in classification procedure is important in two respects. On one hand, output providing alternative classifications and their confidence measures is vital for the operation of the syntax and context analyzer, discussed in Sec. IV of this report. But in addition, it appears that the new procedure will be much easier to design and modify. In the decision-tree approach there was a considerable tendency for all but the most conservative decisions to send characters down the wrong branches of the tree. For example, a seemingly obvious dichotomy is one between characters with enclosures and those without. But many characters have spurious enclosures due to quantization noise, and many actual enclosures are filled in. It is impossible to make even such basic dichotomies without losing a considerable number of characters from their proper branches. If the alternate branches of the decision tree are patched up to handle the characters that fall into them, the program becomes unmanageably complex.

In the confidence-measure approach, however, the decision is made separately for each character category on the basis of all the preprocessed

information. Each case can be decided on the basis of its own merits, so to speak. There is not the pressure to make binary choices like the branching of a decision tree, if there is any significant possibility of losing characters thereby. Furthermore, absolute decisions do not have to be made in any case. The existence of the continuous-valued confidence measure allows a gradual decrease of confidence in a given category as the feature values depart more and more from the values expected for that category. Thus, the natural and beneficial consequence of producing confidence measures for the context analyzer is that the classifier is allowed to express degrees of doubt, as it were, about placing a character in a given category. This situation would seem to mirror the human response to ill-formed text.

The addition to TOPO 2 of a classification routine embodying these concepts is underway. The results of the first preliminary tests are most encouraging. We shall continue to implement the classification routine (and add to the preprocessing as necessary) and report more fully on results in the next report.

The remainder of this section contains the first half of a discussion of the techniques that have been developed for topological preprocessing.

C. Discussion of Topological Preprocessing Operations

Topological features extracted by preprocessing should not only be "natural," but should also meet the allied criterion of "ruggedness." A rugged feature is one whose presence is not changed, and whose characteristics are not greatly altered, by normal variations in the image of a character in a given category. The processing routines used to find the features must be tolerant of variations in the source characters and distortions caused by the scanning process, if they are to produce rugged features. If the image is affected by salt-and-pepper noise, for example, a route to find connected figure components must be able to reject small, isolated figure elements.

The primary feature information concerning a character evidently resides in the strokes forming the character. In fact, if the strokes are defined

as comprising the path(s) that the writing instrument follows in forming the character, it is a tautology that the strokes contain all the feature information. But, in a more practical sense, the stroke information suffers two weaknesses: not all stroke information can be recovered, and other types of features may convey equivalent information in more desirable form.

We may contrast the available stroke information on the hand-printed page with that of "on-line" input to a computer, through, for example, a cathode ray tube and light-pen or a RAND tablet. Two characteristics of on-line input are outstanding. First, time-sequence information and even velocity information about the strokes are available. Second, the strokes are line drawings; they have infinitesimal width. These characteristics make the recognition of characters on-line an entirely different problem from the off-line recognition of characters on a printed page. It is a point of major significance that an off-line printed character must be recognized from its shape alone.

Full stroke information cannot be recovered from an off-line printed character image, owing to the overlapping of strokes in the body of the figure and the masking of the stroke path by the finite width of the stroke. (Thus, it appears that an important quantitative parameter of the difficulty of a handprinted character recognition problem is the ratio of stroke width to character size.) We are led, therefore, to define the stroke information as that information that can be derived from the image by some processing routine, and to look for auxiliary forms of natural, rugged feature information.

Two such feature types are concavities of the figure boundary, and enclosures (holes) within the figure. Others are junctions, or blobs--regions in which strokes come together to form nodes, masses, or areas of confusion. The overall size and location of the character as a whole and of its connected components are important features. We may also add to the list features that are derivable from the stroke information: directions, curvatures, and corners. Finally, the relations among features can be features in themselves, such as the connections of strokes and

the relative placement of strokes, concavities, and enclosures. It appears that the features just listed represent a natural, and profitable, way of presenting handprinted characters.

We turn now to a discussion of the feature types and the computer routines we have used to calculate them (working from a 24 × 24 binary matrix representation of the character images).

1. Figure Extent and Location

The subroutine EXTENT (NFIG, JT, JB, KL, KR) finds the indices of the topmost (JT), bottommost (JB), leftmost (KL), and rightmost (KR) figure points in the image NFIG. (Rows are numbered 1-24 going downward; columns 1-24 from left to right.) NFIG may be a character, a connected component, or any image at all. EXTENT is fast in operation because it need merely scan the image once by rows (computer words) to find the top and bottom of the figure, then scan the word it has collected meanwhile (by OR-ing the rows of the image) to find the left and right boundaries.

The location of a figure is determined by the row and column indices of its center. The center of an object is typically defined as its centroid or center of gravity. Finding the centroid, however, requires a lengthy computation. We prefer the definition

$$JC = (JT + JB)/2$$
$$KC = (KL + KR)/2$$

which locates the center of the smallest rectangle enclosing the figure. This calculation can be performed far faster than finding the centroid. It generally gives values close to the centroid, and may be equally desirable or even preferable for our purpose.

2. Connectivity

A figure is connected if any two of its elements can be joined by a chain of neighboring figure elements. Two definitions of "neighbor," and thus of connectivity, are at hand. We shall call them 4-connectivity and 8-connectivity. In 4-connectivity, the neighbors (N) of an element (X)

are the four adjacent elements vertically and horizontally:

```
 N
NXN
 N
```

In 8-connectivity, the four elements adjacent diagonally are included as neighbors:

```
NNN
NXN
NNN
```

Rosenfeld and Pfaltz describe the two types of connectivity in a recent article.[4] They point out the "paradox," or inelegance, in the connectivity of figure (1) and ground (0) elements related thusly:

```
1 0
0 1
```

The figure and ground are both 8-connected, but neither is 4-connected. The authors fail to remark on the satisfying duality that results from specifying one entity to be governed by 4-connectivity and the other by 8-connectivity, so that only one is connected at a crossover.

It is generally in our interest to maximize figure connectivity, so we choose the figure to be governed by 8-connectivity, and the ground by 4-connectivity. (Often a single marginal figure element will lie diagonally adjacent to the body of the figure, and we can thus avoid having to treat it as a separate figure. Marginal elements and isolated elements due to salt-and-pepper noise can be eliminated by a smoothing operation.[5] Since we seldom receive such noise from the vidicon camera, we avoid the smoothing operation, which represents extra work and is not without some danger of losing significant detail.)

Our choice of figure and ground connectivities means that concavities and enclosures, which are ground areas, will be 4-connected. Thus, in the following image, the figure is connected but the ground is not.

One ground element is an enclosure in the figure.

```
1 1 0 0
1 0 1 0
1 1 0 0
```

3. Subroutines CONN8 and CONN4

Subroutines CONN8 and CONN4 embody the basic connectivity operation. CONN8 works with 8-connectivity; CONN4, with 4-connectivity. Their action is otherwise identical, so only CONN8 will be described. The function of CONN8 is to find those connected components of a figure (FIGA) that include elements of another figure (FIGB). The image composed of the components found is returned by the subroutine as (FIGC). CONN8 has two modes of operation:

```
CONN8   (FIGA, FIGC, FIGB, 0)
              and
CONN8   (FIGA, FIGC, J, K)
```

where J and K range from 1 to 24. In the second mode, the figure in FIGB is taken to consist of a single element located at (J, K).

The operation of CONN8 begins with storing in FIGC an image that is the element-wise logical product of FIGA and FIGB:

```
FIGC (J,K)  =  FIGA (J,K) AND FIBG(J,K)  .
```

This image contains all the 1-bits (usually figure elements) common to FIGA and FIGB. In the second step, an image is formed which includes as 1-bits all the elements of FIGC and all the neighbors of all the elements of FIGC. This image represents the growth of FIGC by one unit over all its perimeter. In the third step the logical product of this "growing" image with FIGA is returned to FIGC, thus restraining the growth of FIGC to elements within FIGA. The second and third steps are repeated until no new elements are generated in FIGC. At this point, FICG has filled out the connected components of FIGA containing elements of the original FIGC.

The elementary operations required by CONN8 are the element-wise logical AND and logical OR functions performed over two 24 × 24 image fields, and operations that shift an image field right, left, up, and down. Such operations are available to us separately in the form of subroutines:

ANDFIG
ORFIG
XORFIG
DIFFIG } (INFIG 1, INFIG 2, OUTFIG) { $(1 \cap 2)$
$(1 \cup 2)$
$(1 \otimes 2)$
$(1 \cap \bar{2})$

and

RSHFIG
LSHFIG
USHFIG
DSHFIG } (INFIG, COUNT, OUTFIG)

For the sake of speed and compactness in CONN8, however, these operations are performed directly by machine-language coding.

The operations just listed are examples of <u>parallel operations</u>, which can be applied in parallel to the elements of an image field (or two) to produce an output image field. The attractiveness of parallel operations in terms of speed is such that entire computers--notably the ILLIAC III at the University of Illinois--have been devised with a bank of processors capable of working in parallel. (Our own 1024-image preprocessor performs a specialized type of parallel operation on the image field.) It should be noted that a conventional computer such as the SDS910 is capable of partial parallel processing by using the logical operations that deal in parallel with the bits of a computer word, representing a row of the image. An operation on a 24 × 24 element field that would require 576 steps sequentially, or one step in a parallel computer, can be performed in 24 steps by the conventional computer, affording a considerable saving in time relative to purely sequential operation.

The CONN8 routine, as it is actually programmed, also takes advantage of the fact that each row of the growing FIGC is immediately available

for the calculation of the next row. This allows the connected region formed in FIGC to cascade in one direction (downward) during the execution of steps 2 and 3, above. If the original FIGB is at or near the top of the appropriate connected component of FIGA, FIGC can be found in very short order. This is a limited example of the sequential processing discussed in the paper by Rosenfeld and Pfaltz.[4]

CONN8 and CONN4 are basic building blocks for other operations described below.

Although we have associated CONN4 with the ground (rather than figure) components of the image, CONN4 is programmed to work on regions composed of elements with the value 1, as does CONN8. Since the figure is normally assigned the value of 1 and the ground the value of 0, a figure-ground complementation is necessary if CONN4 in its present form is to be used on ground regions. This complementation can be performed by the subroutine

CMPFIG (INFIG, OUTFIG).

4. Figure Dissection

An arbitrary figure can be dissected into its 4- or 8-connected components by subroutine

DISE48 (INFIG, KOUNT, NFIGS, MAXNT, MODE).

DISE48 places individual connected components in NFIGS, which is an array of image fields, and returns the component count in KOUNT. DISE48 first searches the input image to find an element with value 1. (This search can be performed by subroutine WNPT, which finds the northmost of the westmost of the figure points.) Either CONN4 or CONN8 is then called, depending on MODE, to find the entire connected component including this element. This component is removed from the input image and placed in the first image field of NFIGS. The process is repeated, filling successive fields of NFIGS, until the input image is exhausted or MAXKNT-1 components have been dissected.

5. Subroutines GROW8 and GROW4

Subroutine GROW8 (INFIG, OUTFIG) expands the figure INFIG by one element in each of the eight major directions. GROW8 performs an operation equivalent to the operation applied to FIGC in the second step of CONN8. GROW8 is useful for finding parts of an image field immediately adjacent to a given area. GROW4 is a routine analogous to GROW8, but involving 4-connectivity.

A routine SHRINK, which strips away the outer layer of a figure, could be devised. SHRINK is, in a rough sense, the inverse of GROW. The two operations are not truly inverse, however, nor are they commutative with each other. For example, the sequence (SHRINK, GROW) eliminates isolated figure points, thus changing the image.

SHRINK can be realized by applying GROW to the complement of the figure to be shrunk, obtained with CMPFIG. Just as there are 4-connected and 8-connected versions of GROW there could be analogous versions of SHRINK.

The GROW and SHRINK operations are quantized analogs of the "grass-fire" method of Dr. Harry Blum of Air Force Cambridge Research Labs.

This description will be resumed in a future report, with a description of routines for finding the perimeter, convex hull, concavities, enclosures, and strokes of a connected figure.

BLANK PAGE

IV INITIAL DEVELOPMENT OF A FORTRAN SYNTAX ANALYZER

A. Introduction

Some brief experiments, described in the Second Quarterly Report, indicated that humans achieve error rates in the range from one to five percent when presented with hand-printed characters in random order. When presented with text material--i.e., printed matter organized into words, sentences, equations, etc.--humans achieve error rates of a small fraction of one percent. Clearly, the human makes use of context in recognizing the individual characters, and it is obvious that a successful FORTRAN text reader will have to do likewise. Accordingly, we undertook the development of a FORTRAN syntax analyzer, which would accept partially mis-identified input from the single-character classifier and produce clean text.

The word "syntax" refers to the formal grammar of the FORTRAN language; hence, the syntax analyzer would make use of the fact that every statement in FORTRAN must obey the rules of FORTRAN grammer. One can also investigate the use of context. The word "context," as opposed to "syntax," refers to the fact that a particular word or character must fit in with the words and characters surrounding it in order for the whole tc make sense. Thus, one could construct a statement in FORTRAN (or in any natural or computer language, for that matter) which obeyed the rules of grammar (syntax) but made no sense because some of the words were meaningless in their particular context. It is anticipated that the text analyzer will eventually make use of contextual information as well, but the current effort emphasizes the grammatical aspects of the FORTRAN language.

B. Structure of the Syntax Analyzer

1. Input to the Analyzer

A pattern classifier is usually thought of as a device that accepts a pattern and decides which class it belongs to. In order to make the most efficient use of the syntax analyzer, however, the classifier will produce not a single decision, but a list of alternative decisions. Moreover, each alternative will be accompanied by a number giving the confidence in that alternative. For example, if the original character (true class) was the letter "O", the classifier might produce the list: ((D 40) (O 30) (Q 10) (Ø 10) (P 10)) meaning that the classifier decided that the character was a "D" with confidence 40, an "O" with confidence 30, a "Q" with confidence 10, the numeral "Ø" with confidence 10, or a "P" with confidence 10. We have called such lists L-lists.

The number of alternatives for any given character to be recognized will vary, depending upon how uncertain the classifier was. If the classifier used is the 9-view piecewise linear machine described in previous reports, for example, then each view might contribute 10% to the total confidence. (Normalization to 90%, 100%, or any other number is immaterial, since the analyzer deals with relative confidence levels.) A single FORTRAN statement would be represented at the input of the syntax analyzer by a list of L-lists, one for each character of the text, where each L-list has the form of the example given above. We have called such lists P-lists. A P-list is the basic input to the syntax analyzer.

2. Breakdown by Statement Types

The FORTRAN language is divided into approximately 35 different statement types. Some of the more common types are the DO statement, arithmetic assignment statement, GO TO statement, and IF statement. The analyzer attempts first to find the statement type that the P-list belongs to, and then calls in a "specialist" program to clean up that statement type and produce the final answer. The determination of the statement type is based on the fact that the syntax of each type, with one exception, requires that the statement begin with a special control word or words. In the examples above, DO, IF, and GO TO are the control

words. The arithmetic assignment statement is the single exception. Thus, the analyzer first finds the average confidence of a match with each of the FORTRAN control words. If the match is sufficiently high with a given control word, then the statement type corresponding to that control word is assumed. If no match is sufficiently high, then the arithmetic assignment statement is assumed. A detailed explanation and theoretical justification for this procedure is given in the Appendix.

3. Specialist Programs

One the statement type has been determined, the specialist program for that type must be called in to produce the final clean FORTRAN statement. We are currently in the process of writing these programs, and to date have completed eight. Of these eight, three are representative of the difficulty we expect to encounter in writing the remainder.

It is difficult to describe these programs in detail without first specifying the syntax of each statement type. Loosely speaking, however, the specialist programs try to break the P-list into small pieces by attempting to find delimiters called for by the syntax. Thus, for example, if the syntax of a given statement type calls for a comma at a certain place, the program will look for the existence of a possible comma, and see if the pieces on each side can be made into the appropriate segments of the statement. If they can be, they are; otherwise, we continue searching for a possible comma. This breakup process can be carried out only to a certain degree of fineness; beyond that point, one must examine a segment of the P-list as an entity, and try to make sense of it. Examples of these "entities" are variable names, numbers (not necessarily single digits), and arithmetic expressions.

At this level in the program we again appeal to the confidence attached to each alternative. For any segment of a P-list, we can find the string of characters it most confidently represents by simply choosing the most confident alternative for each character. Similarly, one could find the second most confident string, third most confident string, etc. Thus, if we arrive at a point where a segment of the P-list must be examined as an entity, we consider the most likely string of

characters, the second most likely, etc., until either a string is found which agrees with the FORTRAN syntax or we are forced to stop because the combinatorial growth in the number of possible strings exceeds our computing power. This process is essentially the same as the method in determining statement type, and the analysis presented in the Appendix applies here as well. The problem of finding the 1st, 2nd, 3rd, ... most confident string of characters is by no means a trivial problem. A solution to this problem involving a modification of the technique known as dynamic programming was proposed by R. E. Larson of SRI and is described in the next section. A program implementing this solution is currently being written.

4. Dynamic Programming

Consider the following P-list that might have been produced by the classifier working on the list of integers 19,8:

```
( ( (/ 60)  (1 30) )
  ( (, 50)  (9 40) )
  ( (, 50)  (7 30)  (9 10) )
  ( (8 40)  (8 30)  (3 20) ) ).
```

This P-list indicates that the first character was classified as a slash with confidence 60 and as a one with confidence 30, etc. By taking the first choice for each of the four characters, we obtain the string /,,8 having the maximum confidence, 200. A brief examination shows that there are two strings having confidence 190--namely /,,8 and /9,8--but even with this simple problem it soon becomes difficult to find all strings of confidence 180, 170, 160, etc.

The dynamic programming solution to this problem uses only the matrix of confidences:

$$C = \begin{bmatrix} 60 & 30 & - \\ 50 & 40 & - \\ 50 & 30 & 10 \\ 40 & 30 & 20 \end{bmatrix} .$$

In general, the ij^{th} element of this matrix is the confidence associated with choosing the j^{th} alternative for the i^{th} member of the string. A selection of a particular string corresponds to a function $j(i)$, and is called a policy, and for each policy there is a total confidence. Our problem is to rank the policies so that if $m < n$ then the total confidence for the n^{th} policy is less than or equal to that for m^{th} policy.

This ranking is accomplished in two steps. First the possible choices for each row of C--i.e., for each stage of the decision process--are considered and the possible partial confidences are systematically recorded. This is done for each stage in succession until all of the possible total confidences have been obtained. Second, the total confidences are considered in succession and all of the possible policies yielding those total confidences are obtained.

The details of this procedure are best described by using our simple example. Consider the possibilities of Stage 1. The first decision yields a partial confidence of 60 and the second yields a partial confidence of 30. These values are recorded as the two lower-left-most nodes of the graph in Fig. 13. Here the numbers by the two lower-most branches indicate which decision was made at Stage 1, and the numbers inside the circled nodes indicate the number of policies that yield the corresponding partial confidences.

Now consider Stage 2. Had we reached the partial confidence of 60 from Stage 1, at Stage 2 the first decision would yield a partial confidence of 60 + 50 = 110 and the second would yield 60 + 40 = 100; on the other hand, had we only reached 30, the results of the Stage 2 decision would be either 80 or 70. All four of these partial confidences obtainable at Stage 2 are shown on the graph.

So far the basic advantage of this approach has not become apparent, since all possible combinations of decisions and partial confidences are exhaustively represented. Were there D possible decisions at each stage, one might fear that an N-stage process would have to show all D^N possible results explicitly. The discrete nature of the process prevents this

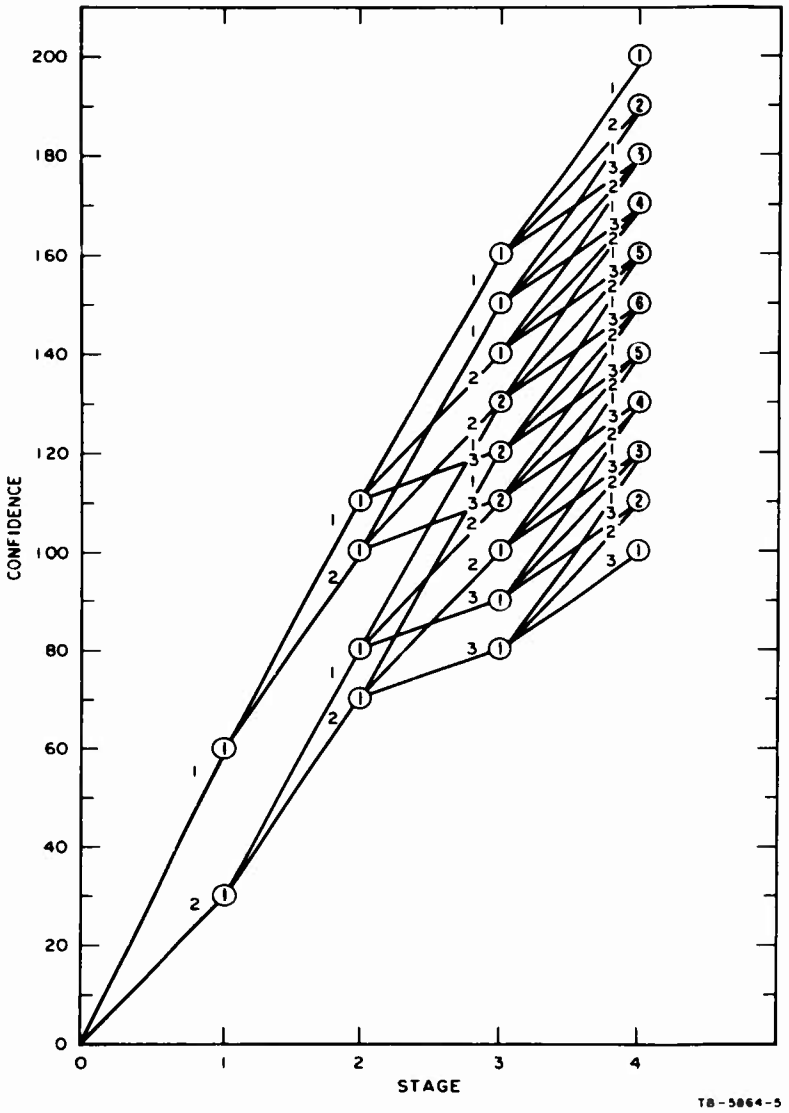

FIG. 13 DYNAMIC PROGRAMMING GRAPH

from happening, however. In particular, note that at Stage 3 there are two ways of obtaining a partial confidence of 130, either by using the policy j(1)=1, j(2)=2, j(3)=2, or the policy j(1)=2, j(2)=1, j(3)=1. Systematic consideration of the partial confidences that can be obtained at Stage 3 finally leads to the complete set of total confidences that can be obtained at Stage 4 shown on the graph. Note that there are only eleven distinct total confidences, which is considerably less than the $2 \times 2 \times 3 \times 3 = 36$ possibilities that could be obtained if all of the possible combinations of choices gave different total confidences. In general, when the individual confidences are integers, there can be no more total confidences than the value of the maximum total confidence, and this usually represents a considerable reduction in computation.

To find all of the policies yielding the highest total confidence, one merely traverses the graph from the corresponding terminal node back to the origin. In this case, there is only one optimal policy: j(4)=1, j(3)=1, j(2)=1, j(1)=1. Two policies yield the next highest total confidence: j(4)=2, j(3)=1, j(2)=1, j(1)=1 and j(4)=1, j(3)=1, j(2)=2, and j(1)=1. Continuing in this way, one obtains the policies in order of descending total confidence, and results down to confidence 150 are listed in Table II.

This example illustrates two important considerations. First, this systematic procedure will indeed yield candidate strings that can be tested for syntactic legality in order of confidence. Thus, for example, the 19th policy yields the string 19,8 which is the syntactically valid integer list having highest confidence. There are several other legal integer lists, such as 1,78 and 1978, but they all have lower total confidence and need not be considered.

Secondly, however, it is clear that this procedure, as it stands, will generate many strings that have no hope of being legal integer lists. Its efficiency could be improved markedly by doing such things as deleting from the P-list any symbols other than digits or commas, retaining in each L-list only the digit having highest confidence, etc., or perhaps even incorporating a dictionary of possible statement labels obtained

Table II

FIRST TWENTY-ONE POLICIES

Total Confidence	Number	Policy				Corresponding String
		j(1)	j(2)	j(3)	j(4)	
200	1	1	1	1	1	/ , , B
190	2	1	1	1	2	/ , , 8
	3	1	2	1	1	/ 9 , B
180	4	1	1	1	3	/ , , 3
	5	1	2	1	2	/ 9 , 8
	6	1	1	2	1	/ , 7 B
170	7	1	2	1	3	/ 9 , 3
	8	1	1	2	2	/ , 7 8
	9	1	2	2	1	/ 9 7 B
	10	2	1	1	1	1 , , B
160	11	1	1	2	3	/ , 7 3
	12	1	2	2	2	/ 9 7 8
	13	2	1	1	2	1 , , 8
	14	1	1	3	1	/ , 9 B
	15	2	2	1	1	1 9 , B
150	16	1	2	2	3	/ 9 7 3
	17	2	1	1	3	1 , , 3
	18	1	1	3	2	/ , 9 8
	19	2	2	1	2	1 9 , 8
	20	1	2	3	1	/ 9 9 B
	21	2	1	2	1	1 , 7 B

irom other parts of the program, and only considering combinations of digits that correspond to possible labels. All of these modifications are specific to integer lists, however, and it would be a digression to discuss them further. The major conclusion is that the dynamic programming technique, together with constraints appropriate to the particular problem, offers a systematic, general method of obtaining the legal string of characters having highest total confidence.

C. Experimental Results

Although the syntax analyzer is far from complete and the final form of the classifier has not yet been determined, we decided to do a small experiment to see if the basic approach of the analyzer were sound. Five handprinted GO TO statements were written and scanned using the TV camera. The patterns were preprocessed using the simulation of the 1024 image preprocessor, and classified using the nine-view piecewise-linear learning machine. Each vote for a class accrued a confidence of 10 for that class. Since the analyzer makes use of spaces in the text and the current scan program does not output spacing information, this information was inserted manually. This is a small point, since we believe that a simple modification of the scan program will enable us to obtain space information with essentially 100% reliability.

The classifier was considered to be incorrect whenever its first--i.e., most confident--choice was in error. On this basis, the classifier made 15 errors out of a total of 34 characters for an error rate of 44%, which is considerably higher than is usual for this classifier. At the output of the syntax analyzer, however, the error rate dropped to 3%.

We reproduce the experimental results for the first statement in their entirety:

Original statement

```
GO  TO 115Ø
```

P-list returned by classifier

```
((( F 40)(E 10) (G 10) (9 10) ( - 10) (5 10))
 (( 0 40)(C 20) (2 10) (Ø 10) ( , 10))
  (( SP   100))
  (( T 60)( - 30))
  (( 3 30)( 0 30)( C 20)(P 10))
  (( SP   100))
  (( 1 60 )( / 30))
  (( 1 70 )( / 20))
  (( 5 90))
  (( P 50)( / 20) ( Ø 10)))
```

First choices of classifier

FO T3 115P

Output of syntax analyzer

GO TO 115Ø

The experimental results for the remaining four statements are summarized below:

Statement 2	GO	TO	26
Classifier's 1st choices	5C	T=	26
Analyzer output	GO	TO	26
Statement 3	GO	TO	988
Classifier's 1st choices	-O	TC	7/8
Analyzer output	GO	TO	788
Statement 4	GO	TO	59
Classifier's 1st choices	KO	FC	59
Analyzer output	GO	TO	59
Statement 5	GO	TO	123
Classifier's 1st choices	GO	TC	/23
Analyzer output	GO	TO	123

D. Conclusion

The experiment just described illustrates dramatically the power of syntax analysis for cleaning up text in which individual characters are misclassified. It is to be recognized, of course, that only a portion of the syntax analysis program has been developed to date, and that correspondingly only a limited specimen of text representing one of the simplest FORTRAN statement types was used in the experiment. We are currently expanding the syntax analyzer, and this activity will continue with the objective of encompassing the several FORTRAN statement types. At a later date we plan to begin incorporating contextual analysis beyond the purely syntactical.

This small experiment also represents a milestone, in that for the first time we have carried actual text from a coding sheet through scanning, preprocessing, classifications, and syntax analysis. Thus we have, in an embryonic sense, demonstrated the complete chain of analysis.

Appendix

A DECISION-THEORETIC FRAMEWORK FOR THE SYNTAX ANALYZER

Appendix

A DECISION-THEORETIC FRAMEWORK FOR THE SYNTAX ANALYZER

In Sec. IV of this report we described the ways in which we are using the syntax of FORTRAN to reduce the error rate in recognizing handprinted text. Abend[6,7] has recently pointed out that compound decision theory provides a natural mathematical framework for incorporating context in pattern recognition. Unfortunately, a rigorous implementation of compound decision theory requires the estimation of too many high-order joint probabilities to warrant considering this approach seriously. Nevertheless, viewing the problem from the vantage point of statistical decision theory serves to clarify the problem and partially to justify our more pragmatic approach.

As a prelude to considering the compound decision problem, consider the problem of classifying a single pattern represented by a set of measurements. These measurements, which might indicate such things as the presence or absence of edges, corners, etc., make up the components of a pattern vector x. For any given "pattern" x, we must pick a category θ, where θ designates one of the possible categories. For the case where x is an allowable FORTRAN character, θ can assume any of 46 different values, corresponding to the 26 letters, 10 numerals, and 10 special characters.

It is well known[8] that in order to obtain a minimum-error-rate classifier, one should compute $p(\theta|x)$ for every possible value of θ, and choose that value of θ for which $p(\theta|x)$ is maximum. By Bayes' rule,

$$p(\theta|x) = \frac{p(x|\theta)\ p(\theta)}{p(x)} \qquad \text{(A-1)}$$

where $p(\theta)$ is the *a priori* probability for θ. Thus the optimum procedure reflects the fact that some characters appear more frequently than others through the presence of $p(\theta)$ in the numerator of Eq. (A-1).

In most of our work to date, we have trained classifiers such as piecewise-linear machines to give as low an error rate as we could obtain. We have tacitly assumed the a priori probabilities $p(\theta)$ to be equal, inasmuch as we have represented each character type with equal frequency in both the training and the testing data. Thus, to the extent that the performance of these classifiers is optimum, we can say that for any pattern x we compute the 46 functions

$$p^*(\theta|x) = \frac{p(x|\theta)\,\frac{1}{46}}{p(x)} \qquad (A\text{-}2)$$

and assign x to the category for which $p^*(\theta|x)$ is maximum.

Consider now the compound decision problem resulting from scanning a syntactically valid FORTRAN statement containing, say, m characters. The result is to obtain a set of m pattern vectors $x_1, x_2, \ldots, x_m$, which can be thought of as the components of another vector, $\underline{x}$. Our problem is to select a corresponding set of categories $\theta_1, \theta_2, \ldots, \theta_m$--i.e., a vector $\underline{\theta}$ such that $p(\underline{\theta}|\underline{x})$ is maximum. Were the θ's statistically independent, we would merely select the best θ_i for each x_i. However, the very reason for considering this problem is that the syntax constraints prevent the θ's from being independent and allow us to obtain a better decision by considering the compound problem than can be obtained from considering each of the component problems separately.

There is one kind of independence assumption we can invoke, however, to obtain a significant simplification of the results. We assume that for any i the i^{th} set of measurements, x_i, depends solely on θ_i, the category of the i^{th} character--i.e., that

$$p(x_i|x_1, \ldots, x_{i-1}, x_{i+1}, \ldots, x_m, \theta_1, \ldots, \theta_m) = p(x_i|\theta_i). \qquad (A\text{-}3)$$

This is actually a very reasonable assumption; it merely states that the types of variations seen in a handprinted character--say, the letter A--depend only upon the fact that it is an A, and not at all upon the fact that it is surrounded by other characters. The fact that certain strings

of characters are illegal, or rarely occur, will be introduced through the a priori probability $p(\underline{\theta})$.

As was stated previously, to obtain a minimum-error-rate classifier, we must compute $p(\underline{\theta}|\underline{x})$ for all possible vectors $\underline{\theta}$ and select the $\underline{\theta}$ for which $p(\underline{\theta}|\underline{x})$ is maximum. By Bayes' rule,

$$p(\underline{\theta}|\underline{x}) = \frac{p(\underline{x}|\underline{\theta})\ p(\underline{\theta})}{p(\underline{x})} \quad . \tag{A-4}$$

It is easy to show by induction that our conditional independence assumption, given by Eq. (A-3), leads to

$$p(\underline{x}|\underline{\theta}) = \prod_{i=1}^{m} p(x_i|\theta_i) \quad . \tag{A-5}$$

Substituting this and Eq. (A-2) into Eq. (A-4) yields

$$p(\underline{\theta}|\underline{x}) = \frac{p(\underline{\theta})}{p(\underline{x})} \prod_{i=1}^{m} [46p(x_i)p^*(\theta_i|x_i)J \quad . \tag{6}$$

This can be simplified further, however, since we are only interested in the variation of $p(\underline{\theta}|\underline{x})$ with $\underline{\theta}$. Thus, if we drop constants and factors dependent solely on $\underline{x}$, we obtain an equivalent compound decision rule by selecting that $\underline{\theta}$ for which

$$q(\underline{\theta}, \underline{x}) = p(\underline{\theta}) \prod_{i=1}^{m} p^*(\theta_i|x_i) \tag{7}$$

is maximum.

Consider now the application of this result to a segment of a FORTRAN statement that we wish to treat as an entity, such as a number or a list of integers separated by commas. Any given $\underline{\theta}$ corresponds to a string of m characters and either is or is not a syntically valid entity. We assume that we are always dealing with valid FORTRAN statements, and hence that $p(\underline{\theta}) = 0$ for invalid entities. Thus we need only consider $\underline{\theta}$'s corresponding to valid strings.

What can we assume about $p(\underline{\theta})$ for valid strings? In some circumstances, it seems reasonable to invoke the principle of insufficient reason and consider all valid strings to be equally likely. For example, if the entity is supposed to be a number, there is little reason to expect one number more than another. In such cases $p(\underline{\theta})$ is a constant factor that does not influence the decision; one merely selects the $\underline{\theta}$ for the valid string for which

$$\prod_{i=1}^{m} p^*(\theta_i | x_i)$$

is maximum. If we define the confidence $c(\theta_i, x_i)$ as the logarithm of $p^*(\theta_i | x_i)$, this is equivalent to selecting the $\underline{\theta}$ for the valid string for which the total confidence

$$\sum_{i=1}^{m} c(\theta_i | x_i)$$

is maximum. Thus the dynamic programming approach described in Sec. IV is applicable in this situation. Strings are considered in order of decreasing confidence until a valid string is encountered, this valid string corresponding to the optimal statistical decision.

There are other common situations, however, in which it is not at all reasonable to consider all valid strings to be equally likely. When dealing with an entire FORTRAN statement, for example, we are much more likely to find the statement starting with a string like DIMENSION or GO TO 913 than XMZQR = VUS. Thus, rather than generating strings of successively lower confidence and rejecting them until we obtain a valid string, we can immediately compute the confidence associated with those strings for which $p(\underline{\theta})$ is known to be large *a priori*--namely, the control words associated with the various statement types. If the string does indeed happen to begin with XMZQR = VUS, then none of these strings is likely to have a high confidence--that is, a confidence that is reasonably close to the maximum possible confidence. On the other hand, if the true string corresponds to any statement type other than an arithmetic assignment statement, we are very likely to obtain a high-confidence

match with the corresponding control word(s) quickly, since we have to consider only one $\underline{\theta}$ for each statement type. Since it is impossible, in practice, to estimate the *a priori* probabilities for all syntactically valid strings, so that some approximate procedure must be used in any case, this appears to be a very reasonable procedure that does not do violence to the guiding ideas of compound decision theory.

ACKNOWLEDGMENTS

The authors would like to express their deep appreciation for the assistance of the following people in the work described in this report: Mr. Michel Henry, a researcher visiting the group from the Centre d'Etudes Nucleaires de Grenoble, France, who developed the corner-detecting masks through many patient hours at the SDS 910 console; Miss Helen Chan, programmer, who has contributed in many ways, notably in writing the CALMMASK program; Mrs. Ann Robinson, programmer, who wrote the ERROR GRAPH program and is assisting the writing of the syntax analyzer; and Mr. Michael Sullivan, technician and programmer, who has performed the continuing task of collecting data from coding sheets with the SCAN 2 program.

REFERENCES

1. M. M. Chodrow, W. A. Bivona, and G. M. Walsh, "A Study of Handprinted Character Recognition Techniques," Technical Report RADC-TR-65-444, Rome Air Development Center, New York (February 1966).

2. W. W. Bledsoe and I. Browning, "Pattern Recognition and Reading by Machine," Proc. Eastern Joint Computer Conference, pp. 225-232 (1959).

3. D. M. Stern and D. W. C. Shen, "Character Recognition by Context-Dependent Transformations," Proc. IEE, Vol. 111, No. 11, pp. 1923-1931 (November 1964).

4. A. Rosenfeld and J. Pfaltz, "Sequential Operations in Digital Picture Processing," J. of the ACM Vol. 13, No. 4 (October 1966).

5. G. P. Dineen, "Programming Pattern Recognition," Proc. Western Joint Computer Conference, pp. 94-100 (1955).

6. Kenneth Abend, "Compound Decision Procedures for Pattern Recognition," Report PR-344, System Sciences Laboratory, Philco, Blue Bell, Pennsylvania (April 1966).

7. Kenneth Abend, "Sequential Compound Decision Procedures for Dependent States of Nature and for Unknown Distribution," Proc. 1966 IEEE Pattern Recognition Workshop (to be published).

8. H. Chernoff and L. D. Moses, Elementary Decision Theory (John Wiley and Sons, New York, 1959).

UNCLASSIFIED

Security Classification

DOCUMENT CONTROL DATA - R & D

(Security classification of title, body of abstract and indexing annotation must be entered when the overall report is classified)

1. ORIGINATING ACTIVITY (Corporate author)	2a. REPORT SECURITY CLASSIFICATION
Stanford Research Institute 333 Ravenswood Avenue Menlo Park, California 94025	UNCLASSIFIED
	2b. GROUP N/A

3. REPORT TITLE

GRAPHICAL-DATA-PROCESSING RESEARCH STUDY AND EXPERIMENTAL INVESTIGATION

4. DESCRIPTIVE NOTES (Type of report and inclusive dates)

Report No. 26, Fourth Quarterly Report - 1 November 1966 to 31 January 1967

5. AUTHOR(S) (First name, middle initial, last name)

Richard O. Duda Peter E. Hart John H. Munson

6. REPORT DATE	7a. TOTAL NO. OF PAGES	7b. NO. OF REFS
March 1967	72	8

8a. CONTRACT OR GRANT NO.	9a. ORIGINATOR'S REPORT NUMBER(S)
DA 28 043 AMC-01901(E)	SRI Project 5864
b. PROJECT NO. IP6-20501 A-448-02-45	
c.	9b. OTHER REPORT NO(S) (Any other numbers that may be assigned this report) ECOM-01901-26
d.	

10. DISTRIBUTION STATEMENT

Distribution of this document is unlimited.

11. SUPPLEMENTARY NOTES	12. SPONSORING MILITARY ACTIVITY
	U.S. Army Electronics Command Fort Monmouth, New Jersey 07703 Attn: AMSEL-NL-A-1

13. ABSTRACT

This report describes the continuing development of preprocessing, classification, and context analysis techniques for hand-printed text, which are advancing at an accelerating pace.

Experiments have been continued with the Piecewise-Linear learning machine, using the outputs of two preprocessors: The PREP 24A simulation of the 1024-image optical preprocessor, and the CALMASK preprocessor, which employs both edge-detecting and corner-detecting masks. A new low test error rate for classification has been achieved on hand-printed alphabets of FORTRAN characters.

Statistics of the performance of the learning machine during a single testing iteration are presented, and shed light on several important questions, such as the distribution of rankings of the desired character category when it is not in first place.

A discussion of the preprocessing methods used in the topological approach to preprocessing and classification is begun.

The initial development of a FORTRAN syntax analyzer is described. A milestone has been reached with the passage of a small sample of actual FORTRAN text from a coding sheet through the scanning, preprocessing, classification, and syntax-analysis programs.

14 KEY WORDS	LINK A		LINK B		LINK C	
	ROLE	WT	ROLE	WT	ROLE	WT
Pattern Recognition						
Adaptive Systems						
Learning Machines						
Preprocessing						
Classification						
Context Analysis						
FORTRAN Syntax Analysis						